

The Pennsylvania State University
The Graduate School
Capital College

Finding a Quota and Set of Weights
to Achieve a Desired Balance of Power
in a Weighted Majority Game

A Master's Paper in
Computer Science
by
Suzanne Rizzo

© 2003 Suzanne Rizzo

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

March 2003

Abstract

This paper describes two algorithms that use the Banzhaf power index to measure the power of voters in a weighted voting system. Given a desired balance of power, these algorithms attempt to calculate a quota and set of weights which produce a Banzhaf power index that is as close as possible to the input. The first algorithm solves the problem by considering quotas and sets of weights for all possible Banzhaf power indices. The second algorithm uses a hybrid genetic algorithm to solve the problem.

Table of Contents

Abstract	i
Table of Contents.....	ii
Acknowledgement	iii
List of Figures.....	iv
List of Tables.....	v
List of Tables.....	v
1 Introduction.....	1
2 The Weighted Majority Game	1
2.1 Terminology.....	1
2.2 Representations of Power	2
2.3 The Banzhaf Index	3
3 Analysis of Algorithm for Calculating Banzhaf Index	5
3.1 Overview of Running Time	5
3.2 Shortcut.....	5
3.3 Running Time Analysis of Algorithm.....	8
4 Reversing the Weighted Majority Game	8
5 Approach #1: Find All Solutions	10
5.1 Correctness of Assumptions	10
5.2 Algorithm.....	12
5.3 Solving the Problem	15
5.4 Results	16
5.5 Example	20
6 Approach #2: Use a Hybrid Genetic Algorithm	22
6.1 Terminology.....	22
6.3 Algorithm.....	25
6.4 Results	26
6.5 Example	27
7 Conclusion	30
References	31

Acknowledgement

I would like to thank Dr. J. Hartzler for his oversight of this project. I would not have been able to complete it successfully without his mathematical insight and guidance. Thanks are also in order for the committee members Dr. T. Bui, Dr. Q. Ding, Dr. P. Naumov, and Dr. L. Null for reviewing this paper. Finally, I am grateful for the support of my husband and classmate, Joe.

List of Figures

Figure 1: Closeness of organisms in population where $q = 51$	28
Figure 2: Closeness of organisms in population where $q = 56$	29
Figure 3: Closeness of organisms in population where $q = 61$	29
Figure 4: Closeness of organisms in population where $q = 65$	30

List of Tables

Table 1: Coalitions of voters for game of size three	3
Table 2: Outcomes of all coalitions for the game [51: 50, 30, 20]	4
Table 3: Pivotal voters for the game [51: 50, 30, 20]	4
Table 4: Coalitions and outcomes for the game [51: 50, 25, 25]	11
Table 5: Pivotal votes for the game [51: 50, 25, 25]	11
Table 6: All Banzhaf indices for games of size two	16
Table 7: All Banzhaf indices for games of size three	17
Table 8: All Banzhaf indices for games of size four	17
Table 9: All Banzhaf indices for games of size five	19
Table 10: Closeness calculation for approach 1 example	21
Table 11: Results for games with known optimal solutions	27
Table 12: Results for games with unknown optimal solutions.....	27

1 Introduction

In this paper, we describe the traditional weighted majority game and the use of the Banzhaf power index to measure voter power. We then explain the reverse game in which we find a quota and set of weights for a desired balance of power. We propose two approaches to solving the reverse game. The first approach is guaranteed to find an optimal solution but has a very large running time. The second approach has a more predictable running time, but may not find the optimal answer. We discuss the results for both approaches and suggest variations for future consideration.

2 The Weighted Majority Game

The weighted majority game describes a particular kind of election. The election is *weighted* because each voter casts a block of votes, rather than exactly one vote. The election is a *majority* because the number of votes needed for a measure to pass must be more than half of the total number of votes.

Applications of the weighted majority game commonly appear in both government and business situations. A typical government election may use a system similar to the United States Electoral College in which each state is given a particular number of votes based on its population. In business, an election may be held in which the stockholder votes are weighted based on the number of shares of stock each holds. An election may be a simple majority in which over 1/2 of the votes are needed for a measure to pass. The election may require more than a simple majority, perhaps over 2/3 or 3/4 of the vote.

2.1 Terminology

A weighted majority game is defined by $[q: w_1, w_2, \dots, w_n]$ where q is the quota (number of votes needed for a measure to pass) and w_i is the weight of the i^{th} voter. Using this definition, the game $[51: 25, 25, 10, 10, 10, 10, 10]$ represents a simple majority game

where 51/100 votes are needed for a measure to pass, and seven voters have weights of either 25 or 10. Likewise, $[7: 5, 3, 2]$ represents a game with three voters and a quota of seven.

In this paper, q and each w_i will be integers ≥ 0 , and q must be at least a simple majority as defined by the following formula:

$$q > \frac{\sum_{i=1}^n w_i}{2}$$

By convention, we list the weights in descending order such that $w_1 \geq w_2 \geq \dots \geq w_n$.

2.2 Representations of Power

Generally, elections are weighted in the interest of fairness. In a business situation, the person who holds the most shares in a company has the most to lose if the company makes a bad decision. Therefore, it is fair for the person with the most shares to have the most input about a decision.

The easiest way to judge the power of the voters is to simply take a straight percentage. If a voter has 20 percent of the votes, it seems as if he would have 20 percent of the power. While this seems logical, a simple example shows that it is not completely accurate. Suppose there is a shareholder who holds 51 percent of the stock of a company. That voter is really in full control of the company because if he votes to pass a motion, the motion passes, regardless of the other shareholders' votes. Likewise, if the controlling shareholder votes to reject a motion, the motion does not pass. If a voter's weight is greater than or equal to the quota, that voter really holds 100 percent of the power.

2.3 The Banzhaf Index

A different measure of power was proposed by John F. Banzhaf III in 1965 [1]. Banzhaf argued that a voter's power is based on his ability to influence the outcome of an election.

To see whether a voter can change the outcome of the election, Banzhaf first lists all the possible coalitions of voters and their votes. A coalition is a set of voters who vote "Yes" in an election. Consider a simple game in which there are three voters. Each voter can vote only two ways – Yes or No. There are 2^3 possible coalitions, shown below, where a "1" represents a "Yes" vote, and a "0" represents a "No" vote.

Coalition #	Coalition	Voter 1	Voter 2	Voter 3
0	{}	0	0	0
1	{V3}	0	0	1
2	{V2}	0	1	0
3	{V2, V3}	0	1	1
4	{V1}	1	0	0
5	{V1, V3}	1	0	1
6	{V1, V2}	1	1	0
7	{V1, V2, V3}	1	1	1

Table 1: Coalitions of voters for game of size three

The second step is to determine whether the vote of each of the coalitions causes the motion to pass or fail. This can be done by adding up the weights of the voters who vote "Yes" and comparing this sum to the quota. If the sum is greater than or equal to the quota, then the motion passes. Otherwise, the motion fails.

Consider the game [51: 50, 30, 20] and the distribution of votes shown above. Here is a chart showing the calculation of the sums, and the outcome of the motion:

Coalition #	Weight 1	Weight 2	Weight 3	Sum(W)	Outcome
0	0	0	0	0	Fail
1	0	0	20	20	Fail
2	0	30	0	30	Fail
3	0	30	20	50	Fail
4	50	0	0	50	Fail
5	50	0	20	70	Pass
6	50	30	0	80	Pass
7	50	30	20	100	Pass

Table 2: Outcomes of all coalitions for the game [51: 50, 30, 20]

The third step is to consider when a voter is pivotal in the outcome. A voter is pivotal if the outcome of the vote would be changed if he (and he alone) changed his vote. Below is a chart indicating whether the voter would change the outcome if he changed his vote:

Coalition #	Pivotal 1	Pivotal 2	Pivotal 3
0	No	No	No
1	Yes	No	No
2	Yes	No	No
3	Yes	No	No
4	No	Yes	Yes
5	Yes	No	Yes
6	Yes	Yes	No
7	Yes	No	No
Count of Pivotal Votes:	6	2	2

Table 3: Pivotal voters for the game [51: 50, 30, 20]

The forth and final step is to add up the number of times each voter is pivotal, and to divide this number by the total number of times any voter is pivotal. We denote the results with a vector called Banzhaf power indices $\beta = (\beta_1, \beta_2, \dots, \beta_n)$. In our example,

$$\beta = \left(\frac{6}{10}, \frac{2}{10}, \frac{2}{10}\right) \text{ or } \beta = (.60, .20, .20).$$

It's interesting to note that in our example, even though Voter #2 has 10 more votes than Voter #3, both have the same ability to change the outcome.

3 Analysis of Algorithm for Calculating Banzhaf Index

The algorithm described in Section 2.3 for calculating the Banzhaf index is an iterative algorithm that involves counting combinations of voter coalitions. Even considering all coalitions is a task requiring exponential time since there are 2^n coalitions, where n is the number of voters.

3.1 Overview of Running Time

Although there are some shortcuts that can speed up the calculations for the Banzhaf index, the algorithm for a single game runs in exponential time. Yasuko and Tomomi Matsui [4] proved that the problem is NP-hard in 2001.

Various improvements to the algorithm have been made to bring the running time down from $O(n2^n)$, but we use the original Banzhaf index calculation in this paper. These improvements include the use of generating functions as implemented in *Mathematica* by P. Tannenbaum [7].

3.2 Shortcut

A symmetry exists in the third step of Section 2.3 which simplifies the calculation of the Banzhaf index. Pivotal votes were defined as votes that ‘changed the outcome’ of the

election. In reality, changing the outcome from Pass to Fail happens exactly as frequently as changing the outcome from Fail to Pass. Therefore, we consider only one of the cases and multiply the result by two to get the actual number of pivotal votes. Following is an explanation of why this shortcut is valid.

Consider two coalitions of voters A and B where every voter votes the same in A and B, except one arbitrary voter i , who votes yes in A and no in B.

An example for a game of size 4 might have the following values:

$i = 2$, the second voter.

$A = \{V1, V2\} = (1,1,0,0)$ where voters 1 & 2 vote yes, and voters 3 & 4 vote no.

$B = \{V1\} = (1,0,0,0)$ where voter 1 votes yes, and voters 2, 3, & 4 vote no.

We will show that A and B are symmetrical: voter i is pivotal in A if and only if voter i is pivotal in B.

First, define Y as the set of voters that are not i who vote yes, and N as the set of voters that are not i who vote no. Because of the way we define A and B, Y and N are the same for games A and B. For the example listed above, $Y = \{1\}$ and $N = \{3, 4\}$.

Suppose voter i is pivotal in A. Since voter i votes yes in A, the measure must pass in A, but without i 's vote, the measure would fail. The following two equations express this:

$$\text{Equation 1: } W(i) + \sum_{y \in Y} W(y) \geq q$$

$$\text{Equation 2: } \sum_{y \in Y} W(y) < q$$

Now suppose voter i is not pivotal in B. That means that the measure either

- 1) fails in B, but would not pass even with i 's vote, or
- 2) passes in B, with or without i 's vote.

However, we know neither of these cases is true because

- 1) Equation 1 states that the measure passes with i 's vote, and

2) Equation 2 states that the measure fails without i's vote.

Therefore, by contradiction, we see that voter i must be pivotal in B if he is pivotal in A.

Now we will show that if voter i is not pivotal in A, he is not pivotal in B, either.

Suppose voter i is not pivotal in A. This can happen in two cases:

CASE 1: The measure fails in A, but even with i's vote, the measure would still fail:

$$\text{Equation 3: } \sum_{y \in Y} W(y) < q$$

$$\text{Equation 4: } W(i) + \sum_{y \in Y} W(y) < q$$

Now suppose voter i is pivotal in B. That means that the measure either

- 1) passes in B, but without i's vote would fail, or
- 2) fails in B, but with i's vote would pass..

However, we know neither of these cases are true because

- 1) Equation 3 states that the measure would fail even without i's vote, and
- 2) Equation 4 states that the measure would fail with i's vote.

CASE 2: The measure passes in A, but even without i's vote, the measure would still pass:

$$\text{Equation 5: } W(i) + \sum_{y \in Y} W(y) \geq q$$

$$\text{Equation 6: } \sum_{y \in Y} W(y) \geq q$$

Now suppose voter i is pivotal in B. That means that the measure either

- 1) passes in B, but without i's vote would fail, or
- 2) fails in B, but with i's vote would pass.

But, we know neither of these cases is true because

- 1) Equation 6 states that the measure would pass without i's vote, and

2) Equation 6 states that the measure would pass without i 's vote.

Therefore, by contradiction, we see that voter i must not be pivotal in B if he is not pivotal in A .

3.3 Running Time Analysis of Algorithm

The following pseudocode shows the breakdown of the algorithm described in section 2.3 above. The running time of the complete algorithm is $O(n2^n)$.

Loop through the 2^n possible distributions of votes	$O(2^n)$
Calculate the outcome of the election based on the distribution	$O(n)$
If the outcome is Pass	$O(1)$
Loop through the n players	$O(n)$
If the player's vote is pivotal in current game	$O(1)$
Count the vote for the player	$O(1)$
End If	
End Loop	
End If	
End Loop	
Loop through the n players	$O(n)$
Divide the player's pivotal votes by the total pivotal votes	$O(1)$
End Loop	

4 Reversing the Weighted Majority Game

The algorithm used to find the Banzhaf indices answers the question of how much power voters in a weighted voting system have. Sometimes, however, the question is reversed. Instead of asking how much power voters have in a given situation, the question is how to create a situation in which the voters have a given balance of power.

In this problem, the inputs and outputs of the weighted majority game are reversed. In the traditional game, the inputs were the quota and set of weights $[q: w_1, w_2, \dots, w_n]$ while the outputs were the Banzhaf power indices $\beta = (\beta_1, \beta_2, \dots, \beta_n)$. In the reverse game, the input is a desired power index $P = (P_1, P_2, \dots, P_n)$ and the outputs are $[q: w_1, w_2, \dots, w_n]$.

The terminology ‘power indices’ is used in the reverse game instead of ‘Banzhaf power indices’ which is used in the traditional game. The distinction between the two is a small but important one: there is a finite number of Banzhaf power indices for a game of a given size, but we do not restrict the input of the reverse game to this finite set.

To illustrate this point, we will consider a game of size 2. We will look at the finite nature of Banzhaf indices later on in Section 5.1, but for now we will think about it intuitively. When we have only two voters, there are only two ways to balance power between them. Either one voter has all of the power $\beta = (1,0)$, or the two voters have equal power $\beta = (.5,.5)$. When we run the reverse game, however, we allow input that is not exactly a Banzhaf index. For instance, we can run the algorithm for $P = (.98,.02)$. In this case, our algorithm will return a quota and set of weights that generates $\beta = (1,0)$ because this Banzhaf index is closer than $\beta = (.5,.5)$.

In the reverse game, the same restrictions apply for q and w_i as in the traditional game:

$$q > \frac{\sum_{i=1}^n w_i}{2} \text{ and } w_1 \geq w_2 \geq \dots \geq w_n.$$

Additionally, the power indices must add up to one: $\sum_{i=1}^n P_i = 1$.

5 Approach #1: Find All Solutions

One way to find the quota and set of weights for a desired balance of power is to find all possible Banzhaf indices for n voters, and choose the one closest to the input. This approach uses the idea that many games are equivalent to each other, so there are only a finite number of Banzhaf indices for a certain number of voters. While this approach is guaranteed to find an optimal solution, the running time makes it an expensive choice.

5.1 Correctness of Assumptions

We have made two broad statements that require further explanation. The first statement is that many games are equivalent to each other. Equivalent games are those that produce the same Banzhaf index even though they have different quotas or a different set of weights. The second statement is that there are a finite number of Banzhaf indices for a game of a given size.

First, we will show that it is possible for two games to be equivalent. Second, we will show that there are a finite number of outcomes. After we show both these statements to be true, it is clear that many games are equivalent.

The game we discussed in Section 2 had the input of $[51: 50, 30, 20]$ and output of $\beta = (.60, .20, .20)$. We will show that a game with the input of $[51: 50, 25, 25]$, also has an outcome of $\beta = (.60, .20, .20)$.

Below is the chart showing whether the measure passes or fails based on every possible coalition of voters:

Coalition #	Votes (0 = No, 1 = Yes)			Weights (based on votes)			Sum (W)	Outcome
	V1	V2	V3	W1	W2	W3		
0	0	0	0	0	0	0	0	Fail
1	0	0	1	0	0	25	25	Fail
2	0	1	0	0	25	0	25	Fail
3	0	1	1	0	25	25	50	Fail
4	1	0	0	50	0	0	50	Fail
5	1	0	1	50	0	25	75	Pass
6	1	1	0	50	25	0	75	Pass
7	1	1	1	50	25	25	100	Pass

Table 4: Coalitions and outcomes for the game [51: 50, 25, 25]

Below is a chart indicating whether the voter would change the outcome if he changed his vote:

Coalition #	Pivotal 1	Pivotal 2	Pivotal 3
0	No	No	No
1	Yes	No	No
2	Yes	No	No
3	Yes	No	No
4	No	Yes	Yes
5	Yes	No	Yes
6	Yes	Yes	No
7	Yes	No	No
Count of Pivotal Votes:	6	2	2

Table 5: Pivotal votes for the game [51: 50, 25, 25]

This is the same chart as listed in section 2 for the game [51: 50, 30, 20], and the Banzhaf index is the same as well: $\beta = (.60, .20, .20)$

Next, we show that there are a finite number of Banzhaf indices for a game of a given size by finding an upper bound on this number. One upper bound can be derived from the Figure 5 above. First of all, there are 2^n possible coalitions of voters. Each voter has two options (to be pivotal, or not to be pivotal) for each of the 2^n coalitions. So there are at most 2^{2^n} possible total outcomes for games of size n .

In reality, there are at least two reasons that the actual number of solutions would be less than the upper bound described above. First, we restrict the game with the two rules 1) q is greater than half the sum of the weights, and 2) the weights are listed in decreasing order from left to right. Secondly, many outcomes would be equivalent because they create fractions that can be reduced. For example, $\beta = (\frac{6}{10}, \frac{2}{10}, \frac{2}{10})$ would be equal to

$$\beta = (\frac{3}{5}, \frac{1}{5}, \frac{1}{5}).$$

5.2 Algorithm

Choosing all games means that we need to run the Banzhaf algorithm for sets of weights that are ordered in all possible ways. To accomplish this, we set an arbitrary upper bound, and choose all sets of weights that are made up of integers between zero and the arbitrary upper bound. For each of the sets of weights, we then choose valid q 's to generate complete games.

To establish the arbitrary upper bound, we use the concept of “spacing”. We need to leave enough “space” between the weights of the games so that we can discover all the possible orderings of weights.

We start by thinking of the weights from right to left. The weight on the far right, W_n is going to be the lowest weight because we list the weights in descending order. The next weight, W_{n-1} can be either equal to W_n , or greater than it. We need to leave 1 space between these weights so that when we assign weights, we can assign either equal weights or unequal weights. We'll denote this spacing between W_{n-1} and W_n by:

$$\text{Space}(W_{n-1}, W_n) = 1$$

Now we'll consider the next weight, W_{n-2} . Like the previous case, we need to leave 1 space so that W_{n-2} can be equal or greater than w_{n-1} . Additionally, we need to leave 1 space so that the weight of the coalition $\{W_n, W_{n-1}\}$ can be less than or equal to W_{n-2} .

$$\text{Space}(W_{n-2}, W_{n-1}) = 2$$

For the next weight, W_{n-3} , we leave 1 space so that it can be equal or greater than W_{n-2} . We leave one space for each of the ${}_3C_2 = 3$ coalitions of size 2: $\{W_n, W_{n-1}\}$, $\{W_n, W_{n-2}\}$, $\{W_{n-1}, W_{n-2}\}$. We leave one space for the ${}_3C_3 = 1$ coalition of size 3: $\{W_n, W_{n-1}, W_{n-2}\}$.

$$\text{Space}(W_{n-3}, W_{n-2}) = 5$$

Similarly, for the next weight, W_{n-4} , we leave 1 space so that it can be equal or greater than W_{n-3} , 1 space for each of the ${}_4C_2 = 6$ coalitions of size 2, 1 space for each of the ${}_4C_3 = 4$ coalitions of size 3, and 1 space for the ${}_4C_4 = 1$ coalition of size 4.

$$\text{Space}(W_{n-4}, W_{n-3}) = 12$$

We are running the algorithm for games of size 2, 3, 4, and 5. For each of these games, n^2 is sufficiently large to use as our arbitrary upper bound. For games of size 2, we need to leave at least one space between the two weights, so looping through weights from 0 to 4 is sufficient. For games of size 3, we need to leave at least 1 space between the last two weights, and at least 2 spaces between the first two weights. Again, n^2 is sufficient

because looping from 0 to 9 allows for this spacing. Likewise, games of size 4 need a total of $1 + 2 + 5 = 8$ spaces between the various weights, and looping from 0 to 16 allows enough spacing. Finally, games of size 5 need $1 + 2 + 5 + 12 = 20$ spaces, and looping from 0 to 25 allows enough spacing.

Now that the arbitrary upper bound has been established, we can look at the remaining algorithm:

- 1) Loop through all legal sets of weights from $(0, \dots, 0)$ to (n^2, \dots, n^2)
- 2) Loop through all partial sums of the weights from w_n to $w_1 + \dots + w_n$
- 3) Assign two values to q for each partial sum x : x and $x-1$
- 4) Run the Banzhaf algorithm for each of the q 's and weights
- 5) If a new Banzhaf index is found, add it to the list

Details of the algorithm:

- 1) Assigning the weights is a simple matter of counting in base n^2+1 . We start with all weights of zero, then begin working our way from right to left, incrementing the weights up to n^2 . For example, size 2 would look something like this: $(0,0), (0,1), (0,2), (0,3), (0,4), (1,0), (1,1), (1,2), (1,3), (1,4), (2,0), \dots, (4,3), (4,4)$
- 2) We use a binary number counting method to find all the partial sums of the weights. These partial sums correspond with the coalitions of voters. For example, in games of size six, we count from 0 to 2^6 and add all the weights whose bits are turned on. If the counter is at 100110, we add the weights $W_1 + W_4 + W_5$ because these weights are turned on with a 1 instead of 0.
- 3) We choose q to be both the partial sum calculated in step 2 above, and one less than this partial sum. This allows us to model games when the partial sum is greater than or equal to q and games where the partial sum is less than q .

4) We run the Banzhaf algorithm on each of the q 's and weights to generate a Banzhaf power index.

5) We add this Banzhaf index to the list if it has not already been found. A Banzhaf Index B is equivalent to a previously found Index B' if it has the same decimal representation for B_1, B_2, \dots, B_n .

Running Time:

- 1) $O(n^{2^n})$ to count in base n^2+1 from all 0's to all 2^n 's
- 2) $O(2^n)$ to count n digits in binary
- 3) $O(1)$ to choose 2 values of q
- 4) $O(n2^n)$ to run the regular Banzhaf algorithm
- 5) $O(2^n n^{2^{n+1}})$ to add an item to the list since it takes $O(n)$ time to check if two Banzhaf Indices are the same, and the size of the list may be as big as $2^n n^{2^n}$.

Total Running Time: $O(n^{2^n})(2^n)(n2^n)(2^n n^{2^{n+1}}) = O(2^{(3n)} n^{2(2n+1)})$

5.3 Solving the Problem

Once all the possible Banzhaf indices have been found for a given size game, we need to solve the actual problem of returning a set of optimal weights for a given Banzhaf index. To do this, we simply loop through the list of Banzhaf indices, and calculate their ‘closeness’ to the input. If $P = (P_1, P_2, \dots, P_n)$ is the input, and $B = (B_1, B_2, \dots, B_n)$ is one of the outcomes, we calculate the closeness of S by $closeness = \sum_{i=1}^n |P_i - B_i|$. We return the solution with the lowest closeness score.

The running time is unaffected by the final step because the selection runs in $O(n \cdot (\text{number of solutions}))$ time. The total running time remains $O(2^{(3n)} n^{2(2n+1)})$.

5.4 Results

All possible Banzhaf outcomes were calculated for games up to size five. Once these possible games were calculated, any Banzhaf Index could be found in a single pass through the solutions in linear time. The running times that are listed were obtained using a PC with a 900MHz Pentium 3 processor. These running times are intended to give a rough idea of the relative time the algorithm takes across various sized games. The times are snapshots of the PC's clock before and after the algorithm was run, and this clock is accurate to the nearest millisecond.

Size 2 Game – 2 possible outcomes

(Running Time = .000 seconds)

Weights		Q	Number of Pivotal Votes			Banzhaf Indices	
W1	W2		V1	V2	Sum	B1	B2
1	0	1	4	0	4	1.000	0.000
1	1	2	2	2	4	0.500	0.500

Table 6: All Banzhaf indices for games of size two

Size 3 Game – 4 possible outcomes

(Running Time = .050 seconds)

Weights			Q	Number of Pivotal Votes				Banzhaf Indices		
W1	W2	W3		V1	V2	V3	Sum	B1	B2	B3
1	1	1	2	4	4	4	12	0.333	0.333	0.333
1	1	0	2	4	4	0	8	0.500	0.500	0.000
2	1	1	3	6	2	2	10	0.600	0.200	0.200
1	0	0	1	8	0	0	8	1.000	0.000	0.000

Table 7: All Banzhaf indices for games of size 3

Size 4 Game – 12 possible outcomes

(Running Time = 3.400 seconds)

Weights				Q	Number of Pivotal Votes					Banzhaf Indices			
W1	W2	W3	W4		V1	V2	V3	V4	Sum	B1	B2	B3	B4
1	1	1	1	3	6	6	6	6	24	0.250	0.250	0.250	0.250
2	2	1	1	4	8	8	4	4	24	0.333	0.333	0.167	0.167
1	1	1	0	2	8	8	8	0	24	0.333	0.333	0.333	0.000
2	2	1	1	5	6	6	2	2	16	0.375	0.375	0.125	0.125
2	1	1	1	4	8	4	4	4	20	0.400	0.200	0.200	0.200
3	2	2	1	5	10	6	6	2	24	0.417	0.250	0.250	0.083
2	1	1	1	3	12	4	4	4	24	0.500	0.167	0.167	0.167
3	2	1	1	5	10	6	2	2	20	0.500	0.300	0.100	0.100
1	1	0	0	2	8	8	0	0	16	0.500	0.500	0.000	0.000
2	1	1	0	3	12	4	4	0	20	0.600	0.200	0.200	0.000
3	1	1	1	4	14	2	2	2	20	0.700	0.100	0.100	0.100
1	0	0	0	1	16	0	0	0	16	1.000	0.000	0.000	0.000

Table 8: All Banzhaf indices for games of size 4

Size 5 Game – 57 possible outcomes

(Running Time = 8 minutes, 10.600 seconds)

Weights					Q	Number of Pivotal Votes						Banzhaf Indices				
W1	W2	W3	W4	W5		V1	V2	V3	V4	V5	Sum	B1	B2	B3	B4	B5
1	1	1	1	1	3	12	12	12	12	12	60	0.2	0.2	0.2	0.2	0.2
2	2	2	1	1	6	10	10	10	6	6	42	0.238	0.238	0.238	0.143	0.143
3	3	2	2	2	7	14	14	10	10	10	58	0.241	0.241	0.172	0.172	0.172
1	1	1	1	0	3	12	12	12	12	0	48	0.25	0.25	0.25	0.25	0
2	2	2	1	1	5	14	14	14	6	6	54	0.259	0.259	0.259	0.111	0.111
3	3	2	2	1	8	12	12	8	8	4	44	0.273	0.273	0.182	0.182	0.091
2	2	2	1	1	7	6	6	6	2	2	22	0.273	0.273	0.273	0.091	0.091
3	3	2	2	1	7	14	14	10	10	2	50	0.28	0.28	0.2	0.2	0.04
4	3	3	2	2	8	16	12	12	8	8	56	0.286	0.214	0.214	0.143	0.143
2	2	1	1	1	4	16	16	8	8	8	56	0.286	0.286	0.143	0.143	0.143
2	1	1	1	1	5	10	6	6	6	6	34	0.294	0.176	0.176	0.176	0.176
3	2	2	2	1	7	14	10	10	10	2	46	0.304	0.217	0.217	0.217	0.043
2	2	1	1	1	5	14	14	6	6	6	46	0.304	0.304	0.13	0.13	0.13
4	3	3	2	1	8	16	12	12	8	4	52	0.308	0.231	0.231	0.154	0.077
3	3	2	1	1	6	16	16	12	4	4	52	0.308	0.308	0.231	0.077	0.077
3	2	2	2	1	6	18	10	10	10	6	54	0.333	0.185	0.185	0.185	0.111
3	2	2	1	1	7	12	8	8	4	4	36	0.333	0.222	0.222	0.111	0.111
4	3	2	2	1	8	16	12	8	8	4	48	0.333	0.25	0.167	0.167	0.083
5	4	3	2	2	9	18	14	10	6	6	54	0.333	0.259	0.185	0.111	0.111
2	2	1	1	0	4	16	16	8	8	0	48	0.333	0.333	0.167	0.167	0
3	3	2	1	1	8	10	10	6	2	2	30	0.333	0.333	0.2	0.067	0.067
1	1	1	0	0	2	16	16	16	0	0	48	0.333	0.333	0.333	0	0
3	2	2	1	1	6	18	10	10	6	6	50	0.36	0.2	0.2	0.12	0.12
5	4	3	2	1	9	18	14	10	6	2	50	0.36	0.28	0.2	0.12	0.04
4	3	3	1	1	7	18	14	14	2	2	50	0.36	0.28	0.28	0.04	0.04
3	3	1	1	1	6	16	16	4	4	4	44	0.364	0.364	0.091	0.091	0.091
4	3	2	2	1	9	14	10	6	6	2	38	0.368	0.263	0.158	0.158	0.053
2	2	1	1	0	5	12	12	4	4	0	32	0.375	0.375	0.125	0.125	0
2	1	1	1	1	4	20	8	8	8	8	52	0.385	0.154	0.154	0.154	0.154
4	3	2	2	1	7	20	12	8	8	4	52	0.385	0.231	0.154	0.154	0.077
3	2	2	1	1	5	20	12	12	4	4	52	0.385	0.231	0.231	0.077	0.077
5	4	2	2	1	9	18	14	6	6	2	46	0.391	0.304	0.13	0.13	0.043

Size 5 game (continued)

Weights					Q	Number of Pivotal Votes						Banzhaf Indices				
W1	W2	W3	W4	W5		V1	V2	V3	V4	V5	Sum	B1	B2	B3	B4	B5
2	1	1	1	0	4	16	8	8	8	0	40	0.4	0.2	0.2	0.2	0
3	2	1	1	1	6	16	12	4	4	4	40	0.4	0.3	0.1	0.1	0.1
3	3	1	1	1	7	14	14	2	2	2	34	0.412	0.412	0.059	0.059	0.059
4	3	2	1	1	7	20	12	8	4	4	48	0.417	0.25	0.167	0.083	0.083
3	2	2	1	0	5	20	12	12	4	0	48	0.417	0.25	0.25	0.083	0
5	3	2	2	1	9	18	10	6	6	2	42	0.429	0.238	0.143	0.143	0.048
3	2	1	1	1	5	22	10	6	6	6	50	0.44	0.2	0.12	0.12	0.12
5	3	3	2	1	8	22	10	10	6	2	50	0.44	0.2	0.2	0.12	0.04
4	2	2	1	1	7	20	8	8	4	4	44	0.455	0.182	0.182	0.091	0.091
4	3	1	1	1	7	18	14	2	2	2	38	0.474	0.368	0.053	0.053	0.053
3	1	1	1	1	5	22	6	6	6	6	46	0.478	0.13	0.13	0.13	0.13
5	3	3	1	1	8	22	10	10	2	2	46	0.478	0.217	0.217	0.043	0.043
2	1	1	1	0	3	24	8	8	8	0	48	0.5	0.167	0.167	0.167	0
4	2	2	1	1	6	24	8	8	4	4	48	0.5	0.167	0.167	0.083	0.083
3	2	1	1	0	5	20	12	4	4	0	40	0.5	0.3	0.1	0.1	0
1	1	0	0	0	2	16	16	0	0	0	32	0.5	0.5	0	0	0
5	3	2	1	1	8	22	10	6	2	2	42	0.524	0.238	0.143	0.048	0.048
4	2	1	1	1	6	24	8	4	4	4	44	0.545	0.182	0.091	0.091	0.091
5	2	2	2	1	7	26	6	6	6	2	46	0.565	0.13	0.13	0.13	0.043
2	1	1	0	0	3	24	8	8	0	0	40	0.6	0.2	0.2	0	0
5	2	2	1	1	7	26	6	6	2	2	42	0.619	0.143	0.143	0.048	0.048
3	1	1	1	1	4	28	4	4	4	4	44	0.636	0.091	0.091	0.091	0.091
3	1	1	1	0	4	28	4	4	4	0	40	0.7	0.1	0.1	0.1	0
4	1	1	1	1	5	30	2	2	2	2	38	0.789	0.053	0.053	0.053	0.053
1	0	0	0	0	1	32	0	0	0	0	32	1	0	0	0	0

Table 9: All Banzhaf indices for games of size 5

5.5 Example

Suppose we are given a Power index of $P=(0.416, 0.287, 0.177, 0.07, 0.05)$. The first step of the solution is to generate all the Banzhaf indices for games of size 5. We've already done this in Table 9, so we'll use these Banzhaf results.

The next step is to calculate the closeness of all the Banzhaf indices using the formula

$$closeness = \sum_{i=1}^n |P_i - B_i|. \text{ This calculation is shown in Table 10 on the next page.}$$

The final step is to choose the solution with the lowest closeness. In Table 10, the solution with the lowest closeness is shown in bold type. The lowest closeness is 0.094, and the Banzhaf index which generated it is $B = (0.417, 0.25, 0.167, 0.083, 0.083)$. The closeness calculation for this solution is expanded below:

$$0.094 = |0.416 - 0.417| + |0.287 - 0.25| + |0.177 - 0.167| + |0.07 - 0.083| + |0.05 - 0.083|$$

The quota and set of weights that generated $B = (0.417, 0.25, 0.167, 0.083, 0.083)$ can be found in Table 9 above in bold type: (7: 4, 3, 2, 1, 1). This quota and set of weights is our solution to the game $P = (0.477, 0.162, 0.158, 0.149, 0.054)$.

Banzhaf Indices					Close-
B1	B2	B3	B4	B5	ness
0.2	0.2	0.2	0.2	0.2	0.606
0.238	0.238	0.238	0.143	0.143	0.454
0.241	0.241	0.172	0.172	0.172	0.45
0.25	0.25	0.25	0.25	0	0.506
0.259	0.259	0.259	0.111	0.111	0.369
0.273	0.273	0.182	0.182	0.091	0.315
0.273	0.273	0.273	0.091	0.091	0.315
0.28	0.28	0.2	0.2	0.04	0.306
0.286	0.214	0.214	0.143	0.143	0.406
0.286	0.286	0.143	0.143	0.143	0.331
0.294	0.176	0.176	0.176	0.176	0.466
0.304	0.217	0.217	0.217	0.043	0.376
0.304	0.304	0.13	0.13	0.13	0.316
0.308	0.231	0.231	0.154	0.077	0.329
0.308	0.308	0.231	0.077	0.077	0.217
0.333	0.185	0.185	0.185	0.111	0.369
0.333	0.222	0.222	0.111	0.111	0.295
0.333	0.25	0.167	0.167	0.083	0.26
0.333	0.259	0.185	0.111	0.111	0.221
0.333	0.333	0.167	0.167	0	0.286
0.333	0.333	0.2	0.067	0.067	0.172
0.333	0.333	0.333	0	0	0.405
0.36	0.2	0.2	0.12	0.12	0.286
0.36	0.28	0.2	0.12	0.04	0.146
0.36	0.28	0.28	0.04	0.04	0.206
0.364	0.364	0.091	0.091	0.091	0.277
0.368	0.263	0.158	0.158	0.053	0.182
0.375	0.375	0.125	0.125	0	0.286
0.385	0.154	0.154	0.154	0.154	0.375

Banzhaf Indices (continued)					Close-
B1	B2	B3	B4	B5	ness
0.385	0.231	0.154	0.154	0.077	0.221
0.385	0.231	0.231	0.077	0.077	0.175
0.391	0.304	0.13	0.13	0.043	0.156
0.4	0.2	0.2	0.2	0	0.306
0.4	0.3	0.1	0.1	0.1	0.186
0.412	0.412	0.059	0.059	0.059	0.267
0.417	0.25	0.167	0.083	0.083	0.094
0.417	0.25	0.25	0.083	0	0.174
0.429	0.238	0.143	0.143	0.048	0.171
0.44	0.2	0.12	0.12	0.12	0.288
0.44	0.2	0.2	0.12	0.04	0.194
0.455	0.182	0.182	0.091	0.091	0.211
0.474	0.368	0.053	0.053	0.053	0.283
0.478	0.13	0.13	0.13	0.13	0.406
0.478	0.217	0.217	0.043	0.043	0.206
0.5	0.167	0.167	0.167	0	0.361
0.5	0.167	0.167	0.083	0.083	0.26
0.5	0.3	0.1	0.1	0	0.254
0.5	0.5	0	0	0	0.594
0.524	0.238	0.143	0.048	0.048	0.215
0.545	0.182	0.091	0.091	0.091	0.382
0.565	0.13	0.13	0.13	0.043	0.42
0.6	0.2	0.2	0	0	0.414
0.619	0.143	0.143	0.048	0.048	0.405
0.636	0.091	0.091	0.091	0.091	0.564
0.7	0.1	0.1	0.1	0	0.628
0.789	0.053	0.053	0.053	0.053	0.751
1	0	0	0	0	1.168

Table 10: Closeness calculation for approach 1 example

6 Approach #2: Use a Hybrid Genetic Algorithm

Another way to find a quota and set of weights for a given balance of power is to use a less time-consuming guess-and-check algorithm. The algorithm we present in this section is not guaranteed to find an optimal solution, but we have seen that it returns very good results.

This approach uses a combination of a genetic algorithm and a greedy algorithm. We will first define some of the terminology used to describe the genetic algorithm. Then we will describe the actual algorithm and the results it achieved.

6.1 Terminology

Genetic algorithms model a problem as a population of organisms that reproduce over time. The algorithm that controls reproduction ensures that the traits of the most fit organisms stay in the population. Hence, when the algorithm is finished, the most fit organism is the solution to the problem.

Organism: An object representing one solution to the problem. In our case, the organism consists of a quota, a set of weights, the calculated Banzhaf index, and a fitness score indicating how far away the Banzhaf index is from the desired Power index.

Fitness: A score indicating the quality of the solution represented by the organism. In our case, the best possible organism is one whose Banzhaf index exactly matches the desired power index. We use the following function to calculate the fitness of an organism: $F = n - \sum_{i=1}^n |P_i - B_i|$ where F is the fitness, n is the number of voters, P is the desired power index, and B is the calculated Banzhaf index of a game. The sum of the absolute values indicates how close the solution is to the desired index. Closeness is best when it is a low number, and fitness is best when it is a high number. Subtracting from n adjusts the fitness accordingly. An organism whose closeness is n would have

every B_i one away from its corresponding P_i . This is the worst possible solution, so its fitness is zero.

Population: A set of organisms. In our case, the population is a set of games.

Generation: A population at a given time during the algorithm.

Crossover: A function that is used to combine two organisms (*parents*) to create a new organism (*child*). In most genetic algorithms, the properties of an organism are encoded in a string known as a *chromosome*. Crossovers combine parts of the encoded strings from both parents to generate a child. Generally, the choice of which genetic material should come from one parent, and which genetic material should come from the other is based on random *cut-points* in the chromosome. At the random cut-point, the genetic material is switched from one parent to the other. In our case, we use a greedy algorithm to combine parents instead of a strict chromosome crossover. We choose two games as the parents (Mom and Dad). We then loop through the individual Banzhaf scores from 1 to n . If Mom's Banzhaf score is best for the i^{th} voter (i.e., if $|P_i - \text{Mom}B_i| < |P_i - \text{Dad}B_i|$) then we choose Mom's weight for the child's i^{th} weight. Otherwise we choose Dad's weight for the i^{th} weight.

Mutation: A random change in the properties of an organism that occurs with a given probability. In our case, when a mutation occurs, we randomly change one of the weights of the game. In order to keep the sum of the weights constant, we also choose a *counterweight* (a different weight used to balance the mutation). We add a random number to the chosen weight, and then subtract the same number from the chosen counterweight.

Reproduction: An algorithm that is run to generate the population of the next generation. In our case, we select the next generation with a combination of greedy choices, crossovers, and mutations.

Parameters: Settings that are assigned at run-time to control the algorithm. We have chosen a fixed set of parameters to use in our tests, but other values might be considered in future tests. The following parameters were used to produce the results listed in section 6.4:

1. **Number of Voters:** 3 to 10

This is determined by the user input.

2. **Sum of Weights:** 100

We use a fixed value so that the weights of every game add up to the same value. This gives us a constant basis for game comparison.

3. **Quota Step:** 5

In order to save time, we do not try every valid quota in our algorithm. Instead, we try only every 5th valid quota. Valid quotas are integers that are greater than half of the sum of the weights.

4. **Maximum Quota:** $2/3$

This parameter is another time-saver. We stop trying quota values when they are more than $2/3$ of the sum of the weights.

5. **Number of Generations:** 5

The number of times a population is produced, either by initialization or reproduction from a previous generation.

6. **Population Size:** 100

The number of organisms in the population of any generation.

7. **Equality Percentage:** 10%

This percentage represents a greedy adjustment we make to the initial population. When we initialize the population, we choose a certain amount of games that have two or more equal weights. This adjustment is made because we found that randomly generating numbers rarely creates games with equal weights.

8. **Parent Percentage:** 10%

The percentage of parents that we choose to be a part of the next generation.

9. **Mutation Percentage:** 10%

The likelihood that a mutation will occur in an organism.

6.3 Algorithm

1. Get input $P = (p_1, p_2, \dots, p_n)$. This is the balance of power we would like to obtain.
2. Loop through a list of quotas. Start with one more than half of the sum of the weights, then try every 5th q up to $2/3$ of the sum of the weights.
3. Initialize the population of the first generation with 100 games using the following three methods:
 - a. Assign the first organism with a greedy method. Set the weights in the same proportion as P for the first game of the population. For example, if we are keeping the sum of the weights equal to 100 and $P = (.5, .25, .25)$ the first organism will have the weights (50, 25, 25)
 - b. Add 10% of the population using games that contain at least 2 equal weights. Randomly select a number x between 2 and n . Assign x of the weights in the game to the same value. Then randomly choose the remaining weights. Sort the weights to preserve the descending order.
 - c. Randomly generate all of the weights for the remaining games. Again, sort the weights to preserve the descending order.
4. Calculate the Banzhaf indices for each of the 100 games in the population.
5. Calculate the fitness for each of the 100 games in the population.
6. Let the population reproduce to create the next using the following methods:
 - a. Choose the best game from the current population to be the first organism in the next generation.
 - b. Choose 10% of the current population to copy into the next generation. Instead of randomly choosing 10% of the games, use a proportional selection scheme so that better games are more likely to be chosen. The probability that a game will be chosen is determined by dividing the game's fitness score by the sum of all the fitness scores in the population.
 - c. Choose the remaining games by crossover. Two different parents are chosen using the same proportional selection scheme as described in (b) above. The crossover operation is run to determine each of the child's weights. If the sum of the child's weights does not add up to the Sum of Weights parameter, an

adjustment is made to one of the weights (chosen randomly). Re-sort the weights to preserve the descending order.

- d. Apply mutations with a given probability of 10%. Loop through all the games in the population we just created. Pick a number between 1 and 100. If the number falls between 1 and 10, the mutation applies. When the mutation applies, we randomly select a weight and counterweight from the game. We then randomly choose an adjustment, add the adjustment to the weight, and subtract it from the counterweight. We shrink the adjustment if it caused a weight to be negative, or greater than the Sum of Weights parameter, and then re-sort the weights to preserve the descending order.
7. Return to step 4, 5 and 6, creating generations until a total of 5 generations have been considered
8. Choose the best game from the 5th generation. If this is the best game that has been considered for all the q, remember it as the best game.
9. Repeat from step 2 until all q have been considered.
10. Return the Banzhaf index of the best game as the solution to the problem.

Running Time:

The running time of this algorithm is dictated by the running time of the Banzhaf algorithm itself, which we showed in Section 3 to have a running time of $O(n2^n)$. We run this algorithm a constant number of times, based on the parameters of the game:

$$\frac{(Sum\ of\ Weights)(Number\ of\ Generations)(Population\ Size)}{2(Quota\ Step)}$$

6.4 Results

Because we generated all of the possible Banzhaf Indices for games up to size 5, we know the optimal answer for a desired power index. Therefore, we can say how many optimal answers were found for games up to size 5.

The hybrid genetic algorithm was run on 1000 games constructed with randomly chosen weights for each of the game sizes listed in the table of results below. It is interesting to note that the optimal answer was found over 97% of the time for all three known game sizes. The running times were obtained using a PC with a 900MHz Pentium 3 processor.

Game Size	Average Closeness	Best Closeness	Worst Closeness	Percent Optimal	Running Time Seconds per Game
3	0.208	0.007	0.398	100	0.558
4	0.141	0.004	0.366	99.9	0.678
5	0.092	0.025	0.297	97.4	0.855

Table 11: Results for games with known optimal solutions

Larger games used a different metric. Since all solutions are not available for these games we simply list closeness and run time.

Game Size	Average Closeness	Best Closeness	Worst Closeness	Running Time Seconds per Game
6	0.061	0.011	0.190	1.211
7	0.053	0.010	0.192	2.154
8	0.050	0.011	0.150	3.882
9	0.049	0.018	0.108	8.157
10	0.050	0.015	0.128	16.437

Table 12: Results for games with unknown optimal solutions

6.5 Example

To illustrate the hybrid genetic algorithm, we will use the same example presented in Section 5.5. In this example, the desired power index was $P = (0.477, 0.162, 0.158, 0.149, 0.054)$. The Banzhaf index that generated the lowest closeness score (0.094) was

$B = (0.417, 0.25, 0.167, 0.083, 0.083)$. One quota and set of weights that generated this Banzhaf index was: (7: 4, 3, 2, 1, 1).

The charts below illustrate the closeness scores of the organisms in each of the five generations. These organisms have been sorted by closeness so that the trend is clear. With each new generation, more organisms have better closeness scores.

The quota and set of weights that was actually generated by this run of the hybrid genetic algorithm is (56: 48, 37, 32, 13, 9, 6). This is one of many solutions that yielded a closeness score of 0.094 and Banzhaf index of $B = (0.417, 0.25, 0.167, 0.083, 0.083)$.

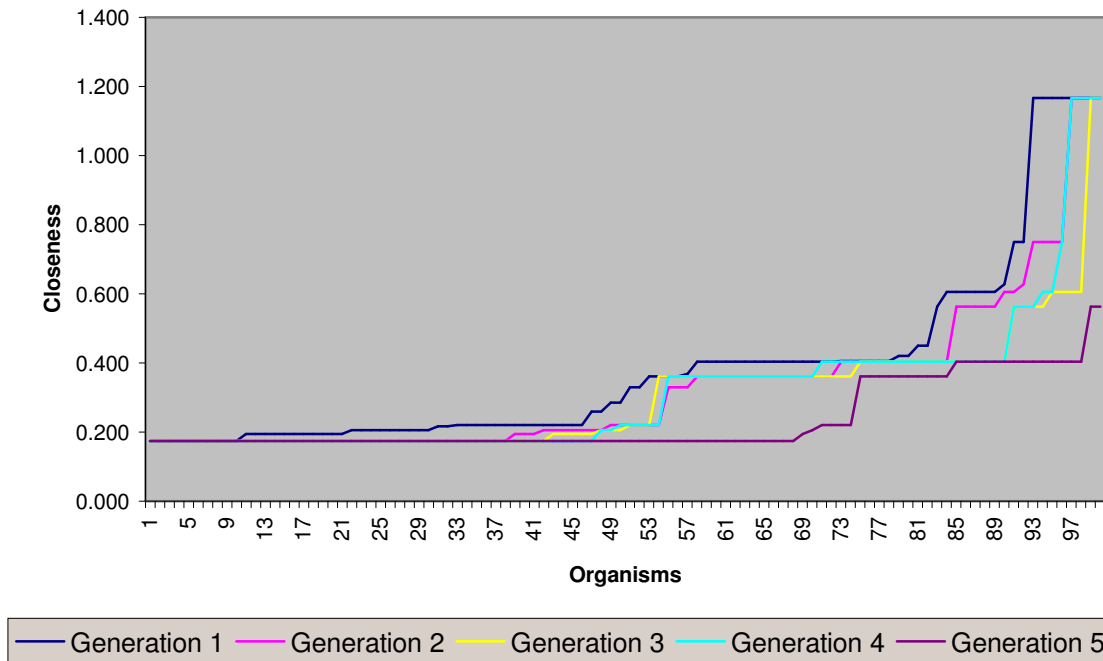


Figure 1: Closeness of organisms in population where $q = 51$

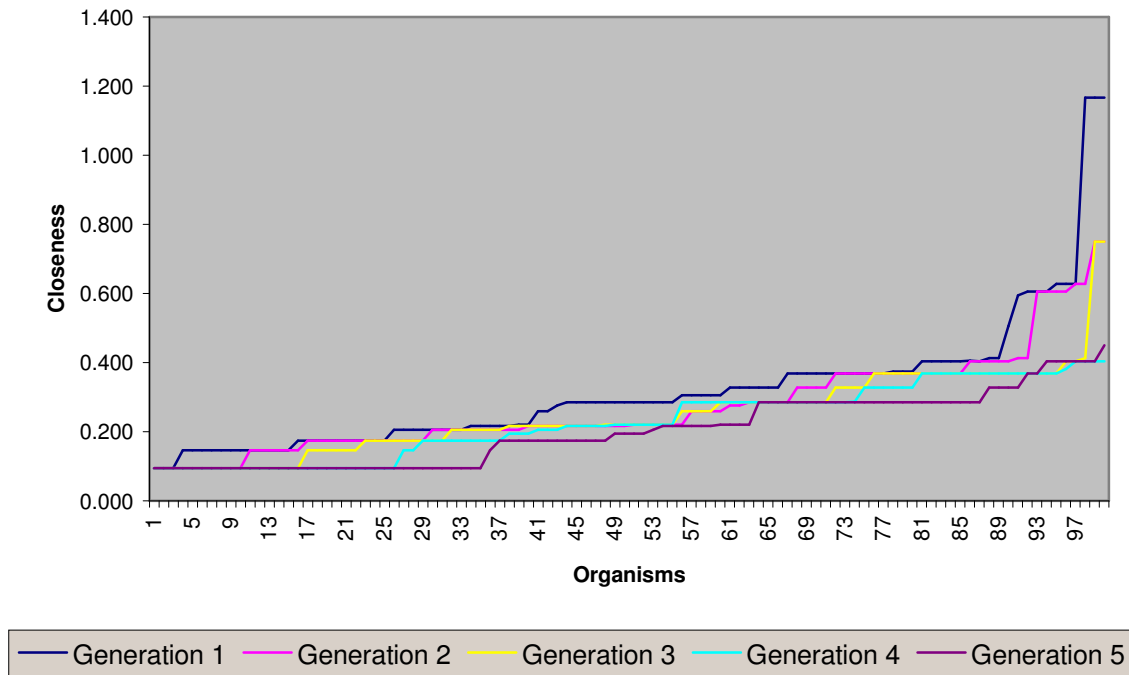


Figure 2: Closeness of organisms in population where $q = 56$

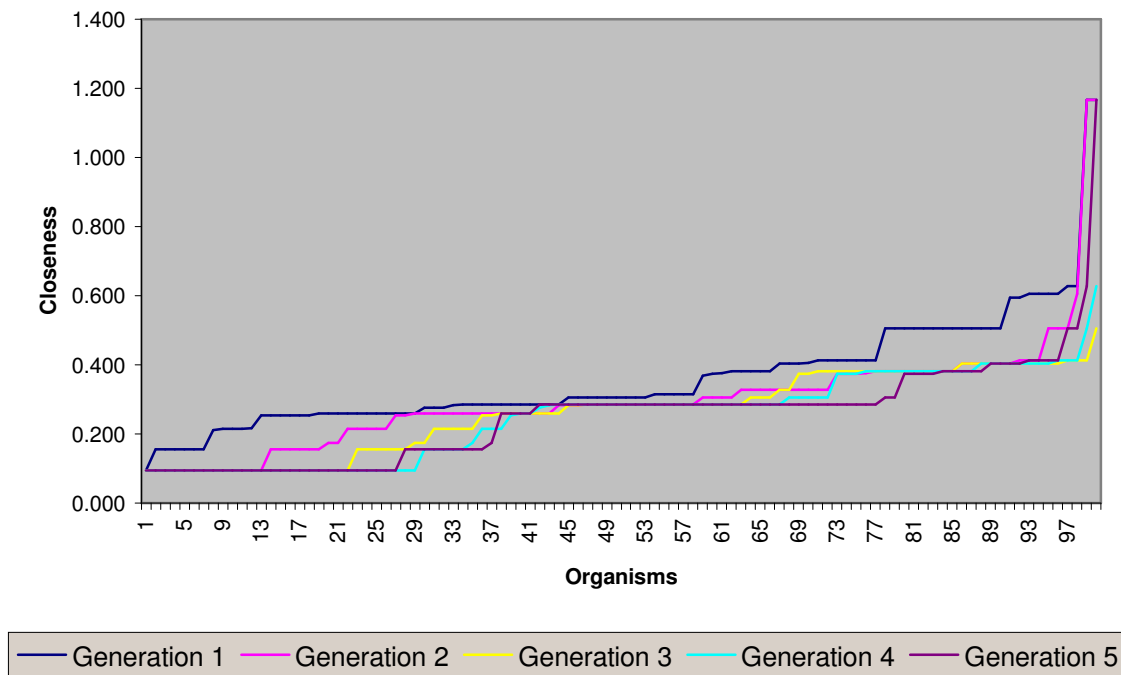


Figure 3: Closeness of organisms in population where $q = 61$

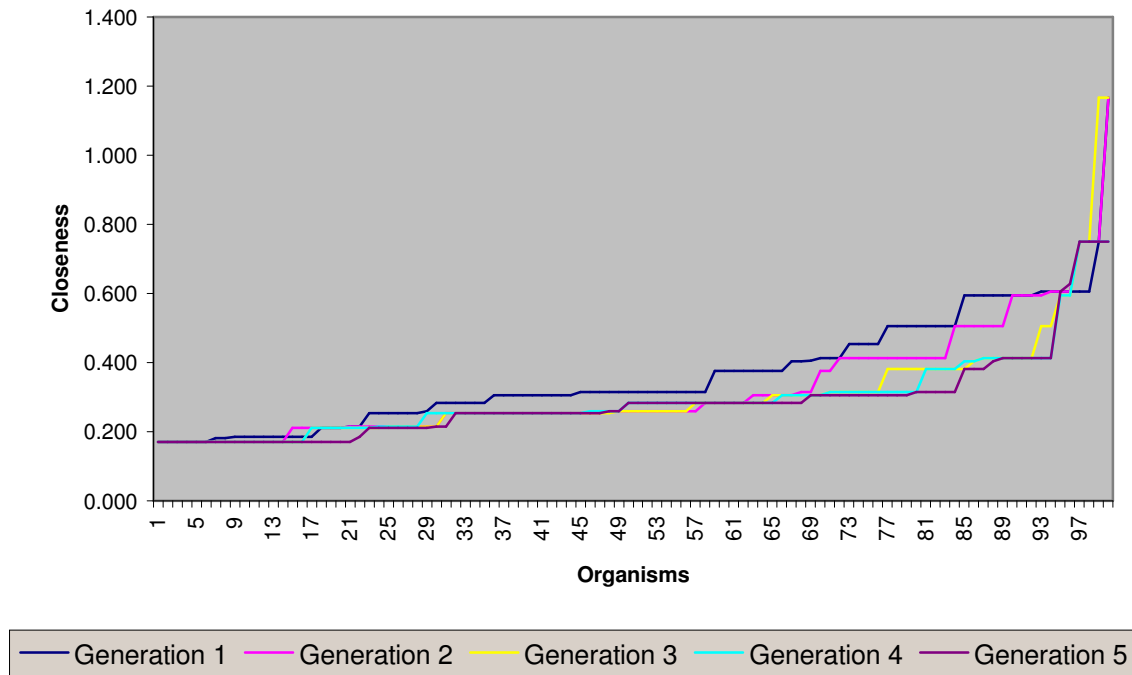


Figure 4: Closeness of organisms in population where $q = 66$

7 Conclusion

In this paper, we proposed two approaches to finding optimal weights and quota for a desired balance of power. The first approach found all solutions for a game of a given size, and then chose the best one. This approach was shown to find an optimal solution for all cases at the expense of a very large running time. The second approach used a hybrid genetic algorithm to find solutions. This approach has a more predictable running time, but was not guaranteed to find the optimal answer.

Future work may include extending the algorithms to work with other power measurements, such as the Shapely-Shubik power index [6]. Also, it would be interesting to take into consideration larger sized games. Modifying the parameters of the hybrid genetic algorithm to change over time, or be dependent on the size of the game is also an option for future work.

References

- [1] J. Banzhaf, 'One Man, 3.312 Votes: A Mathematical Analysis of the Electoral College,' *Villanova Law Review*, 13, 1968, pp. 304-332.
- [2] J.M. Bilbao, J.R. Fernández, A. Jiménez, J.J. López, 'Generating Functions for Computing Power Indices Efficiently,' *Top*, Spanish Statistical and Operations Research Society, 8, 2000, pp. 191-213.
- [3] W. Lucas, 'Measuring Power in Weighted Voting Games', *Case Studies in Applied Mathematics*, Mathematical Association of America, Washington, 1976, pp. 42-106.
- [4] T. Matsui and Y Matsui, "NP-completeness for Calculating Power Indices of Weighted Majority Games", *Theoretical Computer Science*, 263, 2001, pp. 305-310.
- [5] M. Mitchell, "An Introduction to Genetic Algorithms," The MIT Press, Cambridge, MA 1998.
- [6] L. S. Shapley and M. Shubik, "A Method for Evaluating the Distribution of Power in a Committee System", *American Political Science Review*, 48, 1954, pp. 787-792.
- [7] P. Tannenbaum, 'Power in Weighted Voting Systems,' *The Mathematica Journal* 7, 1997, pp. 59-63.